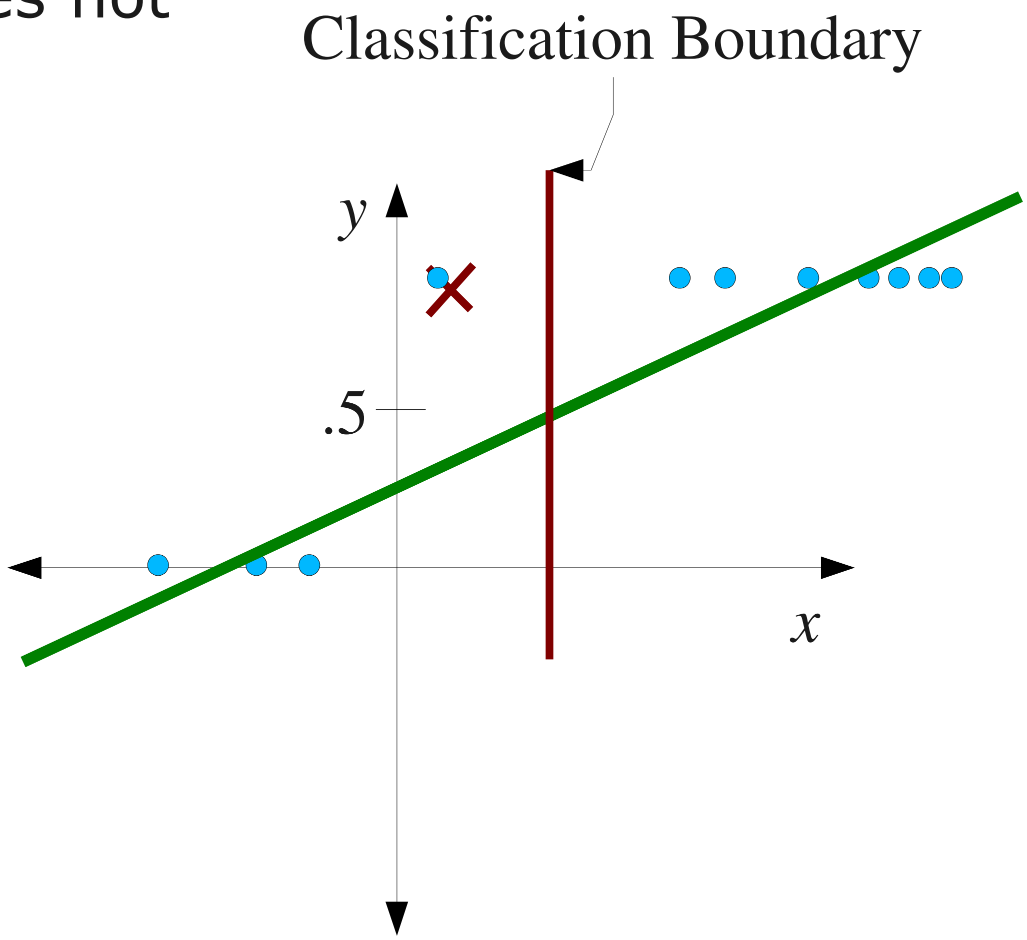
Multi-Layer Neural Networks

Regression vs. Classification

- Now we have the machinery to fit a line (plane, hyperplane) to a set of data points - regression.
- What about classification?
- First thought:
  - For each data point $\boldsymbol{x}$, set the value of y to be 0 or 1, depending on the class
  - Use linear regression to fit the data.
  - During classification assume class 0 if $y < .5$, assume class 1 if $y >= .5$.

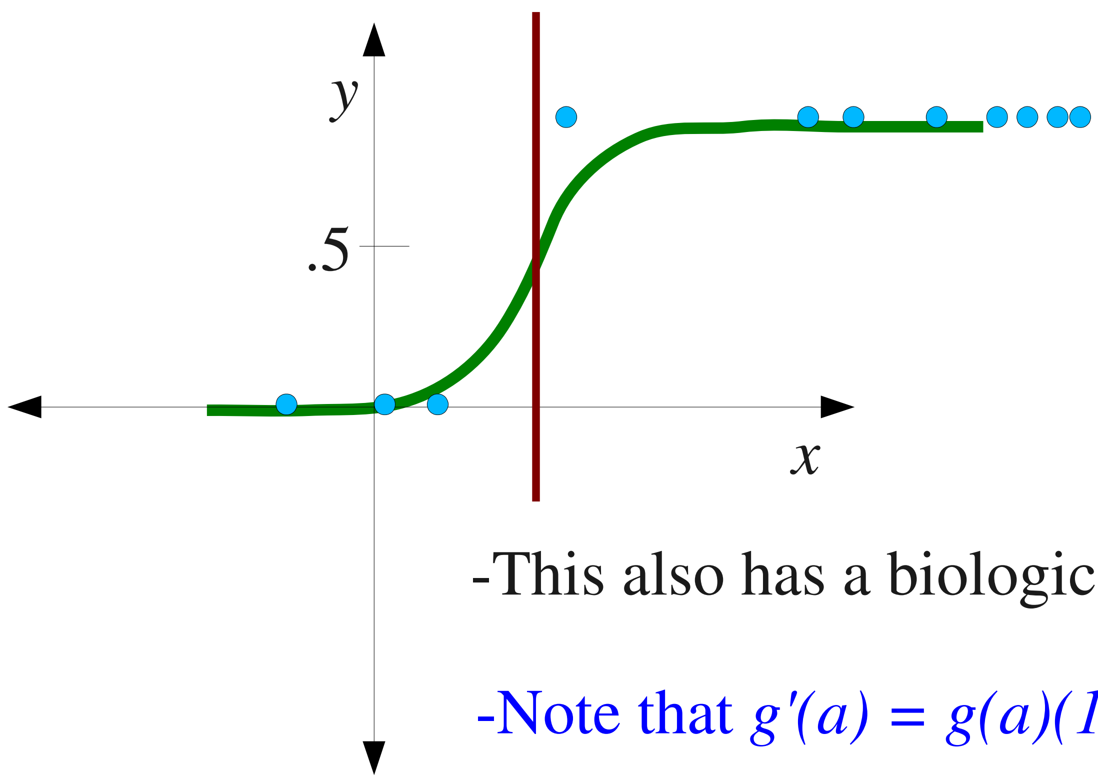
Classification Example

- The least squares fit does not necessarily lead to good classification.



Apply a Sigmoid to the Output

- Let's apply a squashing function to the output of the network: $h(x) = g(\boldsymbol{w}^T \boldsymbol{x})$, where $g(a) = \dfrac{1}{(1+e^{-a})}$



-This also has a biological motivation

-Note that $g'(a) = g(a)(1-g(a))$

The New Update Rule...

- The partial derivative (for a particular example):

$$Error(w) = \frac{1}{2}\left(y - g\left(\boldsymbol{w}^T \boldsymbol{x}\right)\right)^2$$

$$\frac{\partial\, Error(w)}{\partial w_i} = \left(y - g\left(\boldsymbol{w}^T \boldsymbol{x}\right)\right)\frac{\partial}{\partial w_i}\left(\left(y - g\left(\boldsymbol{w}^T \boldsymbol{x}\right)\right)\right)$$

$$= -\left(y - g\left(\boldsymbol{w}^T \boldsymbol{x}\right)\right) g'\left(\boldsymbol{w}^T \boldsymbol{x}\right) x_i$$

- The new update rule: $w_i \leftarrow w_i + \eta\left(y - g\left(\boldsymbol{w}^T \boldsymbol{x}\right)\right) g'\left(\boldsymbol{w}^T \boldsymbol{x}\right) x_i$

- Vector version: $\boldsymbol{w} \leftarrow \boldsymbol{w} + \eta\left(y - g\left(\boldsymbol{w}^T \boldsymbol{x}\right)\right) g'\left(\boldsymbol{w}^T \boldsymbol{x}\right) \boldsymbol{x}$

(This is a lot like "logistic regression" a classical technique from statistics.)

Perceptrons

- Late 50's to mid 60's – Rosenblatt's Perceptrons
  (Original paper: The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain, Psychological Review, 65:386-408)

- Original perceptron formulation used a threshold instead of a sigmoid:

$$g(a) = \begin{cases} 1 \; if \; a > 0 \\ 0 \; if \; a \leq 0 \end{cases}$$

- Learning rule:

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha \left(t - g\left(\boldsymbol{w}^T \boldsymbol{x} \right) \right) \boldsymbol{x}$$

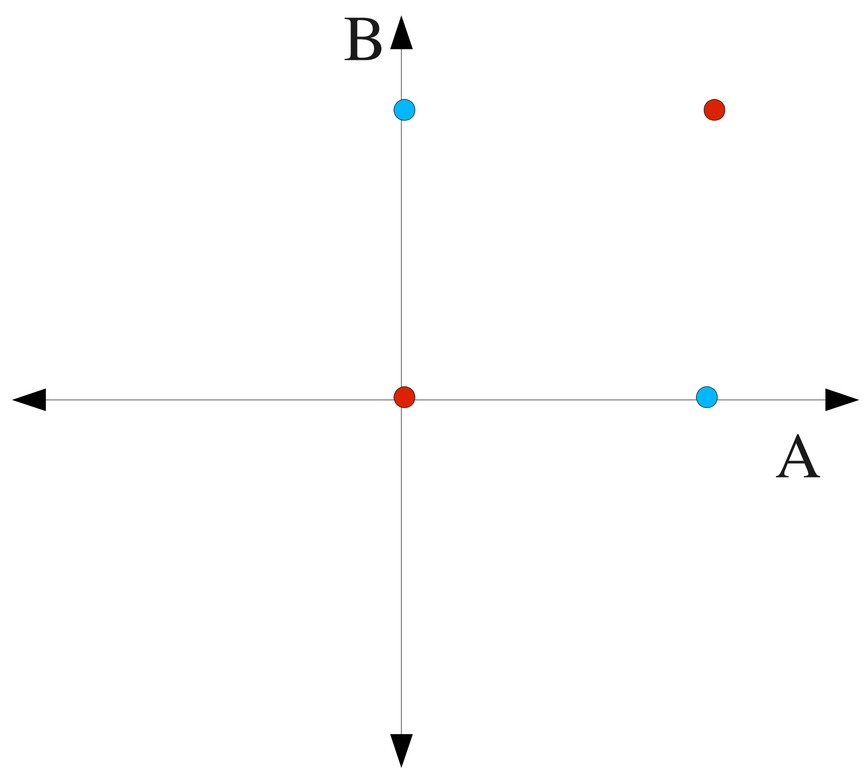
The Rise and Fall of Perceptrons

- 1969 – Minsky and Papert write <u>Perceptrons</u>.
  - Pretty much kills off neural network research.

The Problem...

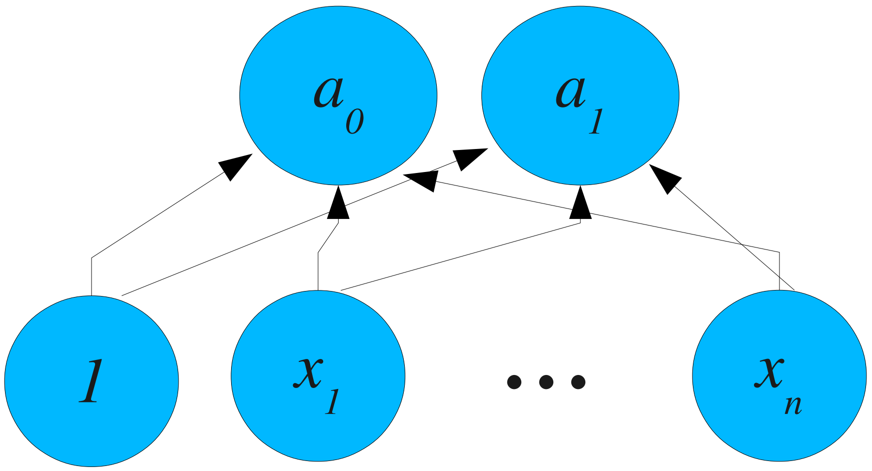
- The perceptron (any single layer neural network) only works if the classes are linearly separable.
- XOR is a problem:

A	B	OUT
0	0	0
0	1	1
1	0	1
1	1	0



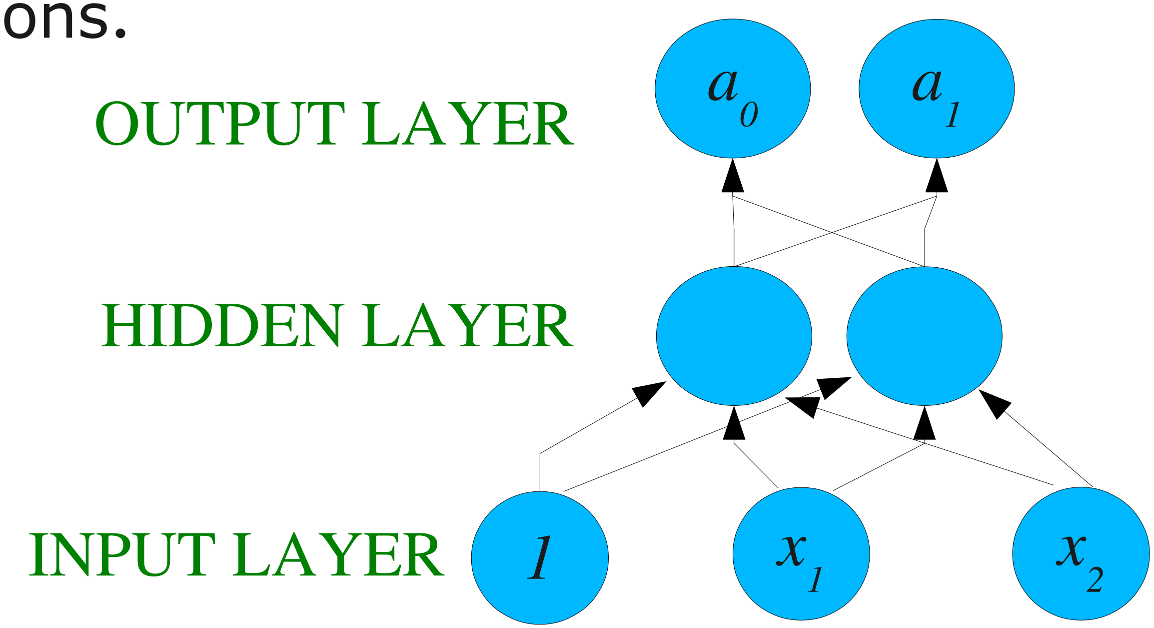
Multiple Output Units

- A network with *M* output units is nothing more than *M* independent perceptrons.



The Solution

- Multi-layer neural networks can represent arbitrary functions.



- We already know how to train the weights at the output layer – this is just a single layer network.
- What about the weights at the hidden layer?

Resurgence of Neural Networks

- Rumelhart, D. E. and J. L. McClelland, Ed. (1986). Parallel distributed processing: Explorations in the microstructure of cognition.
- Backpropagation!

Backpropagation

- Activation at the output layer:

$$a_k = g\left(\sum_j w_{j,k} \;\; g\left(\sum_i v_{i,j} x_i\right)\right)$$

- Here V and W are weight matrices. Units at the input layer are indexed with *i*, hidden with *j* and output with *k.*
- Error metric, assuming multiple output units:

$$Error = \frac{1}{2} \sum_k \left(y_k - a_k\right)^2$$

- Now just compute $\frac{\partial Error}{\partial w_{j,k}}$ and $\frac{\partial Error}{\partial v_{i,j}}$.

Backprop Update Rules

- Let's define a new error term: $\delta_k = Err_k \times g'(in_k)$

- Where: $in_k = \sum_j w_{j,k} a_j$

  $Err_k = (y_k - a_k) = (t_k - g(in_k))$

- So our learning rule becomes:

$$w_{j,k} \leftarrow w_{j,k} + \eta \times \delta_k \times a_j$$

Training the Hidden Layer

- What should the error at the hidden layer be?

$$\delta_j = g'(in_j) \sum_k w_{j,k} \delta_k$$

- This is the backpropagation in backpropagation
- Now the learning rule at the hidden layer is:

$$v_{i,j} \leftarrow v_{i,j} + \alpha \times \delta_j \times x_i$$

The Backprop Learning Algorithm

- Set all weights to small random values
  - **For each training point:
    - FORWARD PASS – compute activations for each layer, starting from the input layer, and working to the output layer.
    - BACKWARD PASS – compute error signals, starting at the output layer and working to the input layer.
    - UPDATE WEIGHTS
  - If the average error is small enough terminate. Otherwise goto **.

Some Issues/Terminology

- Local Minima
- Epoch
- Stochastic Gradient Descent
- Momentum
- Overfitting
- Autoencoding
- Recurrent Neural Networks